Cross-compiling GObject-Introspection

Cambridge Mini-Debconf 2024

COLLABORA

Open First

Hello, world

- Simon McVittie, Senior Software Engineer, Collabora Ltd
  - We do consultancy on open source and open-source-based software
  - I'm currently helping Valve to maintain the Steam Runtime, a Debian derivative
  - Also an upstream maintainer in dbus, Flatpak, bubblewrap, GLib
- smcv, Debian developer
  - GNOME, SDL, Games, Python, Utopia, … teams
  - Technical Committee 2018-2023

COLLABORA

Introduction

G Object what?

- Write one binding for your language
- Get bindings for all GNOME-adjacent libraries
- Dynamic languages: Python, Perl, JavaScript
- Static languages: Rust, C++, Haskell, D, Vala
- Now partially integrated into GLib

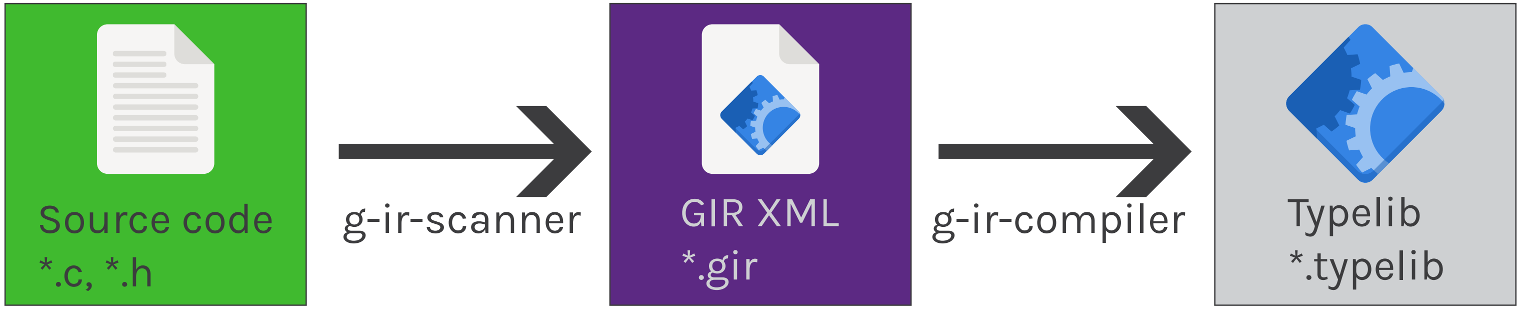
The `GObject` type system

- Object-orientation in C
- Classes, subclasses, objects, virtual methods
- Single inheritance, multiple interfaces (like Java)
- “Boxed” types with a copy function and a free function
- Signals and properties
- Some runtime type information

Two introspection formats

- GIR XML, the API: `Foo-1.0.gir`
  - `libfoo-dev` or sometimes `gir1.2-foo-1.0-dev`
  - Used in static/compiled languages – Rust, C++, Haskell, D, Vala – to generate source
  - XML, human readable, low entropy; human writable (but don't)
  - Architecture independent, except when it isn't
  - Abstract types: `size_t g_variant_get_size (GVariant *)`
- Typelibs, the ABI: `Foo-1.0.typelib`
  - `gir1.2-foo-1.0`
  - Used in dynamic languages – Python, Perl, JavaScript – to call C functions via FFI
  - Dense binary format, architecture dependent
  - Generated from GIR XML with a compiler (and some information loss)
  - Concrete types: `uint64_t g_variant_get_size (GVariant *)`

Generating bindings



That was, in fact, a lie

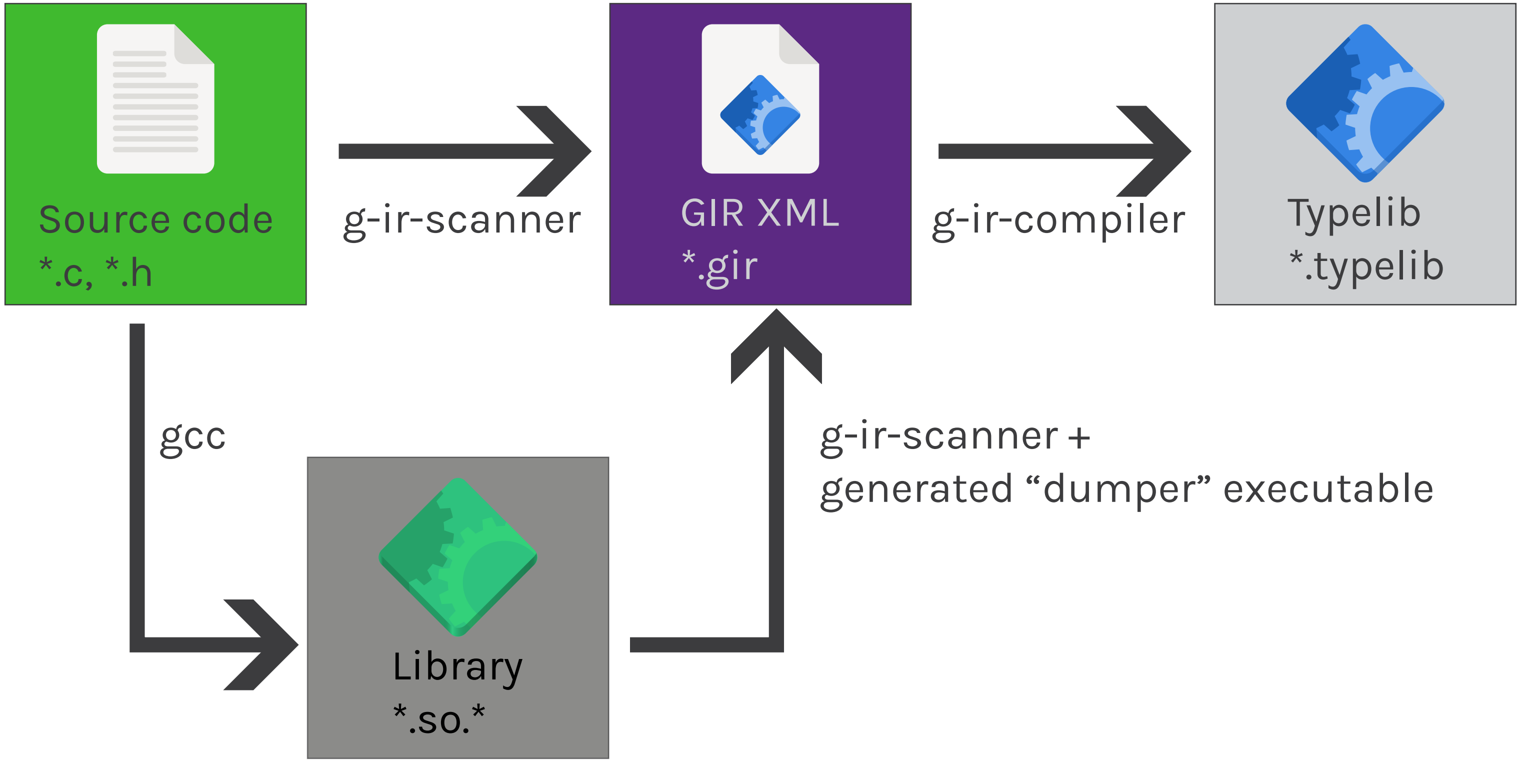
- GObject has run-time type information
- Classes are registered with imperative code
- Architecture-independent, except for when it isn’t

Yes, you can do this

```
if (sizeof (time_t) == 64)
  properties[PROP_TIMESTAMP] = g_param_spec_int64 (...);
else if (sizeof (time_t) == 32)
  properties[PROP_TIMESTAMP] = g_param_spec_int (...);
else
  g_assert_not_reached ();
```

- Please don't

Generating bindings (really)



Cross-compiling

- I have a toolchain on the *build machine*
  - Something fast and/or convenient
  - Could be x86 for example
- I want binaries for the *host machine*
  - Could be some sort of ARM CPU for example
- Some projects use different terms for maximum confusion
  - I'm agreeing with dpkg, Autotools and Meson

The problem

- Compiling typelibs needs an architecture-specific compiler
  - In principle fairly standard, we know how to do this. `arm-linux-gnueabihf-g-ir-compiler`
- Scanning libraries needs to run host-architecture code
  - This is the hard part. I'm running on an x86 (probably), but now I need to run ARM code
- Search paths are different
  - GIR XML is architecture-independent, except when it isn't
  - `/usr/share/gir-1.0` but also `/usr/lib/arm-linux-gnueabihf/gir-1.0`

How to cross-compile G-I, part 1

- Don’t

COLLABORA

How to not cross-compile G-I

How to cross-compile G-I, part 1: don't

- `Build-Profiles: <!nogir>`
- Turn off `gir1.2-foo-1.0`
- Turn off `gir1.2-foo-1.0-dev` if you have it
- Drop GIR XML from `libfoo-dev`
  - This is an API break, be careful
- Drop Vala bindings from `libfoo-dev`?
  - This is another API break, be careful

Compensating for API breaks

- Update providers
  - `libfoo-dev` Provides `gir1.2-foo-1.0-dev` via `${gir:Provides}`
- Update all consumers
  - (Build-)Depends on `gir1.2-foo-1.0-dev`, perhaps via `${gir:Depends}`
- Now you can safely build with `nogir` profile
- Now you can split out `gir1.2-foo-1.0-dev`

OK, but that wasn't the title of this talk

- I did say I was going to talk about cross-compiling

COLLABORA

How to cross-compile G-I

How to cross-compile GI, part 2: really

- Cheat

How `g-ir-scanner` works

- Mostly written in Python
  - Parsing source code
- C extension to interact with `libgirepository`
- Compiles and runs a small C program to learn about `GType`
  - The “dumper”
  - Types, signals, properties, error domains
- Runs `ldd` to learn library dependencies

`g-ir-scanner`, but cross-architecture

- Use the Python code as-is
- Set the search path to use the host `${libdir}`
- Run the dumper binary via `qemu-user`
- Instead of `ldd`, pick apart the ELF header
- Wrapper script: `arm-linux-gnueabihf-g-ir-scanner`

`g-ir-compiler`, but cross-architecture

- We could build 20 cross-compilers
  - 9 Linux release architectures
  - 9 Linux ports (with buildds)
  - 2 ports with non-Linux kernels
  - Needs to “just know” the type sizes and endianness
  - Upstream is unlikely to support this
- Let’s not do that

`g-ir-compiler`, but emulated

- We already need `qemu-user`, right?
- Run the host architecture `g-ir-compiler`
- Good enough! It doesn’t do anything fancy
- Wrapper script: `arm-linux-gnueabihf-g-ir-compiler`

Other tools

- `g-ir-doc-tool`, `g-ir-annotation-tool`
  - Same shape as `g-ir-scanner`, but simpler
- `gi-compile-repository`
  - `g-ir-compiler`, but in GLib
- `gi-decompile-typelib`, `g-ir-generate`
  - Same shape as `g-ir-compiler`
- `gi-inspect-typelib`, `g-ir-inspect`
  - Same shape as `g-ir-compiler`

COLLABORA

Making your build system help

Autotools

- Don’t use `AC_CHECK_PROG`
  - Only looks for `g-ir-compiler`
- Do use `AC_CHECK_TOOL`
  - Looks for `arm-linux-gnueabihf-g-ir-compiler` first
- `introspection.m4` already does the right thing
- That was easy

Meson

- Needs a cross file
  - Or a native file, for non-cross builds
- `${DEB_HOST_GNU_TYPE}-gobject-introspection.ini`
- In future, hopefully `debcrossgen` handles this
- In future, hopefully `meson env2mfile` handles this

CMake

- `/* TODO */`
- Please send a patch or a merge request
  - `gobject-introspection.README.Debian`

Artisanal hand-assembled Makefiles

- If you're lucky, it might use `${CROSS_COMPILE}`?
  - `${CROSS_COMPILE}gcc`
  - `${CROSS_COMPILE}g-ir-compiler`
  - Build with `CROSS_COMPILE=${DEB_HOST_GNU_TYPE}-`

Others

- `/* TODO */`
- Please send a patch or a merge request
  - `gobject-introspection.README.Debian`

COLLABORA

Bootstrapping new architectures

Architecture bootstrapping

- Starting with no packages compiled
  - But we do have a complete build architecture
- Don’t want to rely on qemu
  - It might not even exist

Start small

- Build with `nogir` profile
- No GObject-Introspection tools
- No GIR XML or typelibs
- No tests
- No need for qemu

Complication: `libglib2.0-dev` is too big

- A complete GLib now includes `gi-compile-repository`
- … for the host architecture
- … which is a wrapper script requiring qemu
- … oops

Start small

- `libglib2.0-dev` is now a metapackage
  - Usually still the right build-dependency
  - But avoid it if your package is in the bootstrap set
- Can build-depend on `libgio-2.0-dev` if that's all you need
- You might also need `libgio-2.0-dev-bin`
- You might also need `libglib2.0-bin`

COLLABORA

Side quest: `cross-exe-wrapper`

`cross-exe-wrapper`

- My first prototype wrapper scripts used qemu directly
- Knowing how to run qemu shouldn’t be G-I’s job
- Better: depend on `cross-exe-wrapper`
  - Part of `architecture-properties`, thanks to Helmut Grohne
- Run `${DEB_HOST_GNU_TYPE}-cross-exe-wrapper`
- Does the right thing, whatever that might be

Meson `exe_wrapper`

- You can use this in your Meson builds too
  - `meson setup -Dexe_wrapper=${DEB_HOST_GNU_TYPE}-cross-exe-wrapper`
- In future, maybe `debcrossgen` will handle this
- In future, maybe `meson env2mfile` will handle this

COLLABORA

How can I help?

Existing packages

- Add `Provides: ${gir:Provides}`
  - Or use debhelper compat level 14
- Add `Depends: ${gir:Depends}`
  - Or use debhelper compat level 14

Existing packages

- Build-depend on what you use
  - If your build calls `g-ir-scanner --include=Foo-1.0`
  - Then depend on `gir1.2-foo-1.0-dev`, if it exists
  - If it doesn't exist, send a patch to Foo’s maintainer to Provide it
  - Names are APIs and APIs are names

Existing packages

- Be cross-compile-friendly
  - `file:///usr/share/doc/gobject-introspection/README.Debian.gz`
- If you have a Vala API, give `vapigen` the same treatment
  - https://bugs.debian.org/1061107
  - `src:libportal` has a workaround, but let's not open-code this everywhere

Existing packages

- Implement `nogir`
  - `file:///usr/share/doc/gobject-introspection/README.Debian.gz`
  - Can be done with or without going through the NEW queue

New packages

- Might as well implement `nogir` the nice way
  - If you're going through NEW anyway, have a separate `gir1.2-foo-1.0-dev` package
  - `file:///usr/share/doc/gobject-introspection/README.Debian.gz`

Build systems

- Use Autotools-style GNU-tuple-prefixed cross-tools
  - Yes it's verbose
  - Yes it's a GNUism
  - But it's a de facto standard and it works
- Or centralize the choice of tool in some other way
  - https://bugs.debian.org/1060838
  - https://github.com/mesonbuild/meson/pull/13721

Thank you!

We are hiring

col.la/careers